Edinburgh's Submissions to the 2020 Machine Translation Efficiency Task

Edinburgh's Submissions to the 2020 Machine Translation Efficiency Task

Nikolay Bogoychev[†] **Roman Grundkiewicz**[†] **Alham Fikri Aji**[†] **Maximiliana Behnke**[†]
Kenneth Heafield[†] **Sidharth Kashyap**[‡] **Emmanouil-Ioannis Farsarakis**[‡] **Mateusz Chudyk**[††]

[†]University of Edinburgh
{n.bogoych,rgrundki,a.fikri,maximiliana.behnke,kenneth.heafield}@ed.ac.uk

[‡]Intel Corporation
{sidharth.n.kashyap,manos.farsarakis}@intel.com

[††]Samsung R&D Institute Poland
m.chudyk@samsung.com

Abstract

We participated in all tracks of the Workshop on Neural Generation and Translation 2020 Efficiency Shared Task: single-core CPU, multi-core CPU, and GPU. At the model level, we use teacher-student training with a variety of student sizes, tie embeddings and sometimes layers, use the Simpler Simple Recurrent Unit, and introduce head pruning. On GPUs, we used 16-bit floating-point tensor cores. On CPUs, we customized 8-bit quantization and multiple processes with affinity for the multi-core setting. To reduce model size, we experimented with 4-bit log quantization but use floats at runtime. In the shared task, most of our submissions were Pareto optimal with respect the trade-off between time and quality.

1 Introduction

This paper describes the University of Edinburgh's submissions to the Workshop on Neural Generation and Translation (WNGT) 2020 Efficiency Shared Task[1] using the Marian machine translation toolkit (Junczys-Dowmunt et al., 2018a). The task has GPU, single-core CPU, and multi-core CPU tracks. Our submissions focus on the trade-off between translation quality and speed; we also address model size after submission.

Starting from an ensemble of 4 transformer-big teacher models, we trained a variety of student configurations and on top of that sometimes pruned transformer heads. For the decoding process, we explored the use of lower precision GEMM for both our CPU and GPU submissions. Small models appear to be more sensitive to quantization than large models.

Most of our single-CPU submissions had a memory leak, which also impacted speed; we report results before and after fixing the leak.

[1] https://sites.google.com/view/wngt20/efficiency-task

2 Shared Task Summary

The task measures quality approximated by BLEU (Papineni et al., 2002), speed, model size, Docker image size, and memory consumption of a machine translation system from English to German for the WMT 2019 data condition (Barrault et al., 2019). We did not optimize Docker image size (using stock Ubuntu) or memory consumption (preferring large batches for speed).

The task intentionally did not specify a test set until after submissions were made. This was later revealed to be the average of BLEU from WMT test sets from 2010 through 2019, inclusive. However, the 2012 test set was excluded because it contains English sentences longer than 100 words and participants were promised input would be at most 100 words. We refer to the task's metric as WMT1*. All BLEU scores are reported using sacrebleu.[2]

The CPU tracks used an Intel Xeon Platinum 8275CL while the GPU track used an NVIDIA T4. For speed, the official input has 1 million lines of text with 15,048,961 space-separated words.

3 Teacher-student training

Following Junczys-Dowmunt et al. (2018b) and Kim et al. (2019), all our optimized models are students created using interpolated sequence-level knowledge distillation (Kim and Rush, 2016), and trained on data generated from a teacher system.

Teacher We used the sentence-level English-German system from Microsoft's constrained submission to the WMT'19 News Translation Task (Junczys-Dowmunt, 2019). It is an ensemble of four deep transformer-big models (Vaswani et al., 2017), each with 12 blocks of layers in encoder and decoder, model size of 1024, filter size of 4096,

[2] BLEU+case.mixed+lang.en-de+numrefs.1+smooth.exp+test.wmt*+tok.13a+version.1.4.8 for various WMT test sets.

Proceedings of the 4th Workshop on Neural Generation and Translation (WNGT 2020), pages 218–224

www.aclweb.org/anthology/D19-56%2d

Model	Emb.	FFN	Enc./Dec. Depth	Voc.	Params.	Size	BLEU WMT16	BLEU WMT19	BLEU WMT1*
Teacher ×4	1024	4096	12/12	32k	385.5M	1.5GB	42.4	42.5	36.4
Large	1024	3072	6/6 tied	32k	108.4M	414MB	41.0	43.0	35.3
Base	512	2048	6/2 tied	32k	39.0M	149MB	40.0	42.7	34.6
Tiny.Untied	256	1536	6/2	32k	16.9M	65MB	39.0	42.1	33.4
Tiny	256	1536	6/2 tied	32k	15.7M	61MB	38.7	41.5	33.0
Tiny.8k	256	1536	6/2 tied	8k	9.6M	37MB	37.4	40.6	31.7
Micro.8k	256	1024	4/2 tied	8k	7.0M	27MB	35.7	38.8	30.0

Table 1: Architectures and reference BLEU scores (on a GPU) for the teacher and student models. Reported values are: size of embedding and filter layers, the number of encoder/decoder layers, vocabulary size, the total number of parameters, and model size on disk. WMT1* is defined in Section 2.

and 8 transformer heads.[3] The ensemble achieved 42.5 BLEU on the official WMT19 test set when decoded with beam size of 8. We refer the reader to the original paper for more details on how this system has been built.

Data and training Our student models were trained on pairs of original source and teacher-translated target sentences generated from parallel English-German datasets and English News Crawl data available for WMT19 (Barrault et al., 2019). For parallel data, we generated 8-best lists and selected translations with the highest sentence-level BLEU to reference sentences. Monolingual data was translated with beam size of 4. We filtered the data with language identification using FastText[4] (Joulin et al., 2017), and then scored all sentence pairs with a German-English transformer-base model trained on a subset of original parallel data, about 7 million sentences. The obtained log probabilities were normalized with $\exp(0.1 \cdot p)$ and used for data weighting during training. We also removed ca. 5% of examples with worst scores from each dataset, except Paracrawl (Bañón et al., 2020), from which we used only 15M sentences with highest scores for processing. This procedure is similar to the single-direction step of the dual cross-entropy filtering method (Junczys-Dowmunt, 2018). The final training set consisted of 185M sentences, including 20M of originally parallel data.

All student models were trained using the concatenated English-German WMT test sets from 2016–2018 as a validation set[5] until BLEU has stopped improving for 20 consecutive validations, and select model checkpoints with highest BLEU scores. Since a student model should mimic the teacher as closely as possible, we did not use regularization like dropout and label smoothing. Other training hyperparameters were Marian defaults for training a transformer-base model.[6]

Student models All our students have standard transformer encoders (Vaswani et al., 2017) and light-weight RNN-based decoders with Simpler Simple Recurrent Unit (SSRU) (Kim et al., 2019), and differ in number of encoder and decoder blocks, and sizes of embedding and filter layers. Most models use shared vocabulary with 32,000 subword units created with SentencePiece (Kudo and Richardson, 2018), but we also experimented with a smaller vocabulary with only 8,000 units for model size optimized systems. Used student architectures are summarized in Table 1.

Interestingly, our student models do much better with originally English input, resulting in generally higher BLEU on the WMT19 test set w.r.t. the teacher's performance than on test sets from previous years, which consist of both translations and translationese. For example, the teacher achieves 42.4 and 42.2 BLEU on originally English and originally German subsets of the WMT16 test set, respectively, while the Base student model has 42.5 and only 35.6 BLEU. We think the reason for this is that student models were trained solely on teacher-translated data without back-translations.

4 Attention pruning

Attention is one of the most expensive operations in the transformer architecture, yet many of the heads can be pruned after training (Voita et al., 2019). Moreover, the lottery ticket hypothesis (Frankle

[3] This system refers to the (4×c) configuration in Table 2 from the original paper.

[4] https://fasttext.cc/blog/2017/10/02/blog-post.html

[5] The validation sentences were not teacher-translated.

[6] Available via `--task transformer-base`.

Model	Enc. heads	Params.	Size	BLEU WMT19	BLEU WMT1*	WPS
Tiny	8 8 8 8 8 8	15.7M	61MB	41.5	32.9	2050
Tiny.Steady.i12	2 0 1 2 3 4	14.5M	56MB	41.1	32.4	2282
Tiny.Steady.i14	0 0 1 1 1 3	14.3M	55MB	40.8	32.1	2350
Tiny.Pushy.i6	2 2 2 2 2 2	14.5M	56MB	41.4	32.4	2298
Tiny.Pushy.i7	1 1 1 1 1 1	14.3M	55MB	40.2	31.5	2346

Table 2: Students with pruned encoder attention. Words per second (WPS) is evaluated in *float32* with a single CPU core on the official input (Section 2).

and Carbin, 2018) and subsequent work on pruning optimisation (Frankle et al., 2019) suggests that pruning is less damaging during training rather than after training. Hence we combine these two ideas to prune attention heads during training.

Since we are starting from a relatively optimized model (Tiny in Table 1) whose decoder has one tied layer with SSRU self-attention, our pruning approach focuses on the 48 encoder heads. We apply a late resetting strategy that iteratively removes heads in short training loops (Frankle et al., 2019). This method starts by training the full model for 25k batches to create a checkpoint. Then we repeatedly train for 15k updates, remove N heads and revert the rest of the parameters to their value from the aforementioned checkpoint. Inspired by Voita et al. (2019), we calculate attention "confidence". Each time a head appears, we take the maximum of its attention weights. These maximums are then averaged across all appearances of the head to form a confidence score. Attention heads with high confidence are considered to contribute the most to the overall network performance. Thus, we remove the N least confident heads in each pruning iteration.

We try removing $N = 3$ or $N = 6$ heads per iteration, dubbing these Steady and Pushy in system names, respectively. Since the algorithm usually picks one head from each layer, the final architecture differs. For example, removing 6 heads per iteration results in a monotonic attention distribution across the 6 encoder layers. For submissions, we pruned 36 of the 48 heads; as an additional experiment we tried removing 42 of the 48 heads. The final attention distribution, size and BLEU scores for those models are presented in Table 2.

Considering that our students perform better on newer testsets, the pruning results show that it is possible to remove at least 75% of self-attention heads in an encoder with an average 0.4 BLEU loss. With harsher pruning, the model with even numbers of heads performs better than the one missing any from the first two layers. This indicates that, in extreme cases, it is better to have at least one head per layer than none. Since the dimension of each head was small (256 / 8 = 32), pruning has not reduced the overall size of the models drastically. The speed-up is about 10% on CPU with 75% encoder heads removed. In terms of on GPU, our best pruned model gains 15% speed-up w.r.t. words per second (WPS) losing 0.1 BLEU in comparison to an unpruned model (Tab. 4).

5 CPU optimizations

For our CPU optimization we build upon last year submission (Kim et al., 2019). We use the same lexical shortlist, but we extend the usage of 8bit integer quantized GEMM operations to also cover the shortlisted output layer in order to have faster computation and even smaller model size.

5.1 8-bit quantization

Quantization from 32-bit floats to 8-bit integers is well known (Kim et al., 2019; Bhandare et al., 2019; Rodriguez et al., 2018) and reportedly has minimal quality impact. For this year's submission, we used *intgemm*[7] instead of *FBGEMM*[8] as our 8bit GEMM backend. Vocabulary shortlisting entails selecting columns from the output matrix and *intgemm* can directly extract columns in its packed format. The packed format reduces memory accesses during multiplication. Users can also specify arbitrary postprocessing of the output matrix while it is still in registers before writing to RAM. Currently we use this to add the bias term in a streaming fashion, saving a memory roundtrip on the common $A * B + bias$ operation in neural network inference; in the future we plan to integrate activation functions.

[7] https://github.com/kpu/intgemm/
[8] https://github.com/pytorch/FBGEMM

Model	Size	*xzip* Size	BLEU WMT19	BLEU WMT1*	Words per second 1 core Leak	1 core Fixed	Multi-core Fixed
Base + float32	149MB	135.0MB	42.6	34.53	843	843	19849
+ 8bit-untuned	38MB	25.4MB	42.5	34.29	Out of RAM	1648	39100
+ log-4bit	19MB	15.8MB	42.3	34.10		Run as *float32*	
Tiny + float32	65MB	58.9MB	41.5	32.91	2220	2220	47030
+ 8bit	17MB	11.2MB	41.6	32.89	1028	3135	70037
+ log-4bit	8MB	6.7MB	40.0	31.46		Run as *float32*	
Tiny.8k + float32	37MB	33.4MB	40.6	31.70	1956	1956	42085
+ 8bit	9MB	7.0MB	39.5	30.61	Not submitted	2664	60011
+ log-4bit	5MB	4.1MB	37.5	28.51		Run as *float32*	
Micro.8k + float32	27MB	21.6MB	38.8	30.03	2459	2459	53204
+ 8bit	7MB	4.4MB	37.5	29.01	2094	3229	79992

Table 3: Model sizes, average BLEU scores and speed for quantized models. For the official submission we only used the 8-bit quantized models. More information about the unquantized models can be found in Table 1. The suffix "-untuned" means the model was quantized without continued training. In the multi-core setting, fixing the memory leak had minor impact on speed so we only report fixed numbers. Here, size excludes a 315 KB sentence piece model and an optional (but useful for speed) 11 MB lexical shortlisting file.

Last year (Kim et al., 2019), parameters were quantized and packed offline from a fully trained model. This year, we noticed quality degradation when quantizing smaller models and therefore introduced continued training. Continued training ran for 5000–7000 mini-batches, emulating 8-bit GEMM by quantizing the activations and weights then restoring them to 32-bit values, borrowing from methods used for 4-bit quantization (Aji and Heafield, 2019).

Quantization entails computing a scaling factor to collapse the range of values to $[-127, 127]$. For parameters, this scaling factor is computed offline using the maximum absolute value[9] but activation tensors change at runtime. This year, we changed from computing a dynamic scaling factor on the fly for activations to computing a static scaling factor offline. We decoded the WMT16 dataset and recorded the scaling factor $\alpha(A_i) = 127/max(|A_i|)$ for each instance A_i of an activation tensor A. Then, for production, we fixed the scaling factor for activation tensor A to the mean scaling factor plus 1.1 standard deviation: $\alpha(A) = \mu(\{\alpha(A_i)\}) + 1.1 * \sigma(\{\alpha(A_i)\})$. These scaling factors were baked into the model file so that statistics were not computed at runtime.

All parameter matrices are prepared either offline, or when decoding the first word (in the case of the output layer) and later on they are reused for the GEMM operations (or in the case of the output layers, columns associated with vocabulary items are extracted from the prepared matrix).

For the GEMM operations at the attention layer, we used *cblas_sgemm_batched* from Intel's *MKL* Library. Model sizes, translation quality and speed are reported in Table 3.[10]

Memory leak Most of our CPU submissions had a memory leak due to failing to clear a cache of shortlisted output matrices. Hence our official CPU submissions using intgemm had unreasonable memory consumption after translating 1 million lines as specified in the shared task. In one case, this exceeded 192 GB RAM on the c5.metal instance and a submission was disqualified; in other cases the submissions ran but used too much RAM and likely more CPU time as a consequence. In practise, the negative effect on speed was only evident in the single core submissions because multi-core submissions divided work across processes.

5.2 Log 4-bit quantization

Model parameters follow normal distribution: most of them are near-zero. Therefore, a fixed-point quantization mechanism such as in Section 5.1 is not suitable when quantizing to lower precision. We can achieve a better model size compression by using a logarithmic 4-bit quantization (Aji and Heafield, 2019).

We start by quantizing a baseline model into 4-bit precision. We leave the biases unquantized as they do not follow the same distribution as the rest

[9] We tried a variety of statistics, including minimizing mean squared error, but none worked as well as continued training.

[10] Code for these models is at https://github.com/marian-nmt/marian-dev/tree/intgemm_reintegrated_computestats

of the parameters matrices and therefore quantize poorly. Moreover, the compression rate is practically unaffected since the biases are small in terms of number of parameters. Finally, the model must be fine tuned under 4-bit precision to restore the quality lost by quantization.

With 4-bit precision, we can achieve around 8x model size reduction. While 4-bit log quantization is in principle hardware-friendly since it uses only adds and shifts, current CPUs and GPUs do not natively support it (GPUs do support 4-bit fixed-point quantization, but this reduced quality compared to log quantization). The additional instructions required to implement 4-bit arithmetic made inference slower than with native 8-bit operations. Therefore, we focus on model size, useful for downloading, and dequantize before running the model in *float32*.

Model sizes and BLEU scores are reported in Table 3. Generally, quantizing the model is a better choice when aiming for lower model size, compared to reducing model parameters. For example, Base + log-4bit is as small as 19MB, while losing just 0.4 BLEU compared to the baseline. In contrast, the Tiny model is 65MB, but loses 1.5 BLEU compared to the *float32* and the *int8* settings.

We see that 4-bit log quantization achieves the best size and performance trade-off. For example, our Base + log-4bit (19MB) achieves the highest average BLEU of 34.1 among other models of similar size, such as Tiny + 8bit (17MB, 32.89 BLEU). Similarly, Our Tiny + log-4bit (8MB) achieves an average BLEU of 31.46, compared to others with similar range, for example Micro.8k + 8bit (9MB, 30.61 BLEU). However, larger models are more robust towards extreme quantization, compared to smaller models. Our Tiny.8k + log-4bit degrades significantly in terms of quality.

5.3 Multi-core configuration

For the multi-core track, we swept configurations of multiple processes and threads, settling on 24 processes with 2 threads each. The input text is simply split into 24 pieces and parallelized over processes. The mini-batch sizes did not impact performance substantially and 32 was chosen as the mini-batch size. The code profile under *VTune* revealed that the performance was limited by memory bandwidth, hence, the Hyperthreads available on the platform were not put into use and the 48 cores were saturated using 24 processes (Tange, 2011) running 2 threads each. Each process was bound to two cores assigned sequentially and to the memory domain corresponding to the socket with those cores using *numactl*. Output from the data-parallel run is then stitched together to produce the final translation.

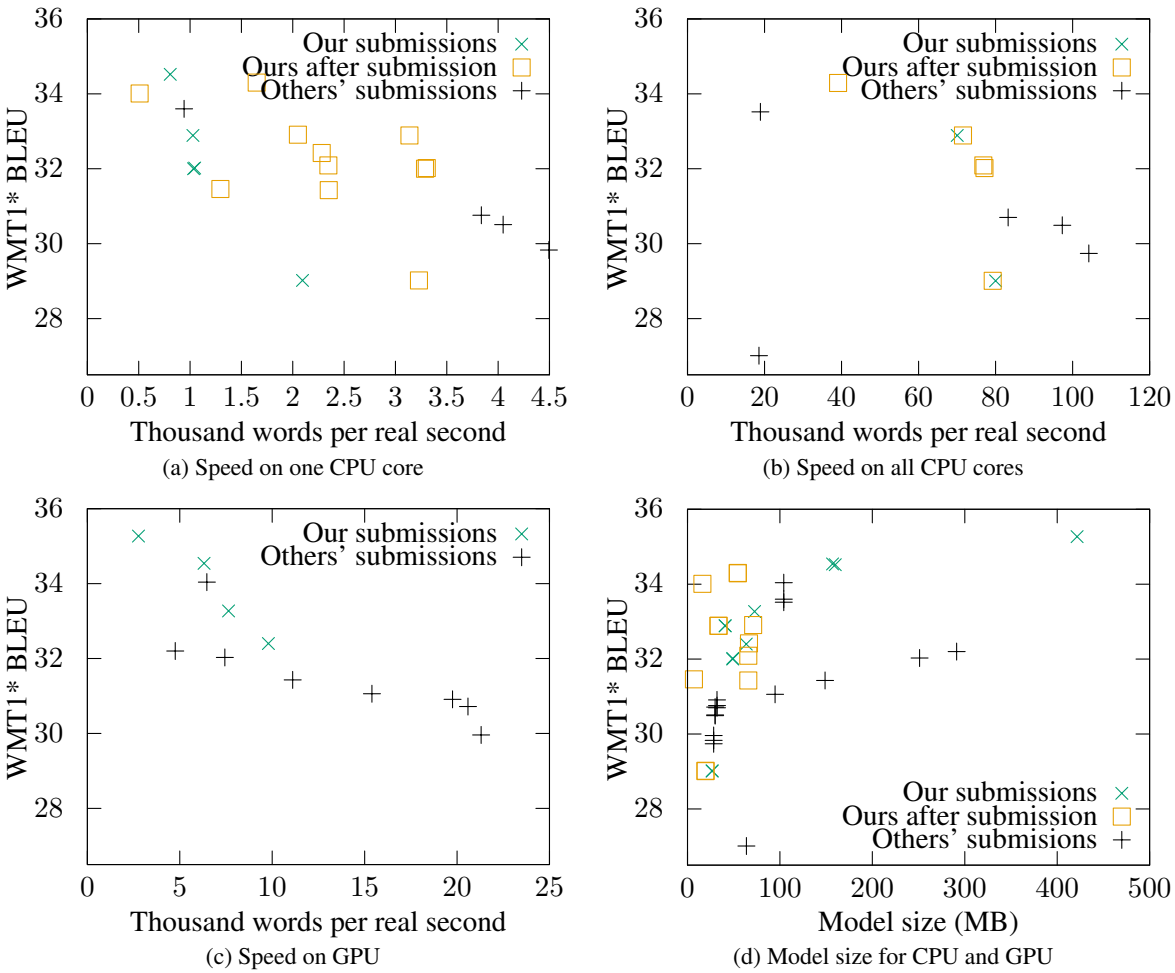
Model	BLEU WMT19	BLEU WMT1*	WPS
Large ($\star$)	43.0	35.27	2748
- w/o 16-bit	43.0	35.29	1764
- w/o shortlist	43.0	35.29	1775
Base ($\star$)	42.7	34.54	6138
Tiny.Untied ($\star$)	41.9	33.27	7602
Tiny	41.5	32.90	8210
Tiny.Steady.i12	41.4	32.36	9518
Tiny.Pushy.i6 ($\star$)	41.0	32.40	9508

Table 4: Performance of student models measured on an AWS *g4dn.xlarge* instance with one NVidia T4 GPU. BLEU scores, total translation times, and word per seconds (WPS). Models with ($\star$) have been submitted to the GPU track.

6 GPU systems

This year, we did not implement any GPU-specific optimizations and focused on comparing the performance of student architectures, developed for CPU decoding, on the GPU. We made 4 submissions to the GPU track. The results for all student models, averaged across 3 runs are reported in Table 4. We decode on GPU using batched translation with mini-batch of 256 sentences, pervasive FP16 inference, and lexical shortlists (Kim et al., 2019). These are features already available in Marian 1.9.

The average speed-up from decoding in 16-bit floats is 21%, depending on the model architecture. The larger the model size, the larger speed improvement, with as high as 56% improvement for the Large student model, through 32% for Base, and only 13-18% for Tiny models. This is with barely any change in BLEU, lower than $\pm$0.1. Models with pruned transformer heads are faster than the original Tiny model by 15% on GPU, but decrease the accuracy by 0.1-0.5 BLEU on the WMT19 test set. On this relatively small data set, we notice a small translation speed decrease of up to 2% from using lexical shortlists. Running concurrent streams on a single GPU did not yield significant improvements for us.



(a) Speed on one CPU core

(b) Speed on all CPU cores

(c) Speed on GPU

(d) Model size for CPU and GPU

Figure 1: Performance of our models compared to other teams. Not all models sought to optimize both speed and space. For example, models stored in 4 bits ran with *float32*.

7 Results and discussion

All submissions and select experiments are depicted in Figure 1.

We explored a variety of ways to optimize the trade-off between quality, speed, and model size. We use an ensemble of 4 transformer-big teacher models to train a number of different student configurations. Smaller student models are faster to decode, but also further degrade the performance compared to the ensemble of teachers. Furthermore, we apply gradual transformer head pruning to the student models. While pruning the number of heads does not reduce the number of parameters significantly, it has a major impact on the computational cost and is beneficial for increasing translation speed, at a small penalty in BLEU score.

On the software side, we experiment with a number of methods that reduce the precision for the GEMM operations. For our GPU submissions, we decode using 16-bit floats and for CPU ones we use 8-bit integers. We note that the smaller (in terms of number of parameters) the model is, the more impacted quality is by quantization, and the bigger the model is, the larger the speed increase is. We found that fine tuning with a quantized GEMM can recover some of the quality loss from quantization.

We also experimented with logarithmic 4-bit model compression, which did not yield increased translation speed due to hardware, but produced the smallest model sizes.

Acknowledgements

We would like to thank Marcin Junczys-Dowmunt for sharing his English-German WMT'19 NMT system that we used as a teacher for our experiments.

This work was supported by funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825303 (Bergamot) and by the Connecting Europe Facility (CEF) - Telecommunications from the project No 2019-EU-IA-0045 (User-focused Marian).

This work was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) op-

erated by the University of Cambridge Research Computing Service (http://www.csd3.cam.ac.uk/), provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/P020259/1), and DiRAC funding from the Science and Technology Facilities Council (www.dirac.ac.uk).

References

Alham Fikri Aji and Kenneth Heafield. 2019. Neural machine translation with 4-bit precision and beyond. arXiv preprint arXiv:1909.06091.

Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. 2019. Findings of the 2019 conference on machine translation (wmt19). In Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1), pages 1–61, Florence, Italy. Association for Computational Linguistics.

Marta Bañón, Pinzhen Chen, Barry Haddow, Kenneth Heafield, Hieu Hoang, Miquel Esplà-Gomis, Mikel L. Forcada, Amir Kamran, Faheem Kirefu, Philipp Koehn, Sergio Ortiz Rojas, Leopoldo Pla Sempere, Gema Ramírez-Sánchez, Elsa Sarrías, Marek Strelec, Brian Thompson, William Waites, Dion Wiggins, and Jaume Zaragoza. 2020. ParaCrawl: web-scale acquisition of parallel corpora. In Proceedings of the 2020 Annual Conference of the Association for Computational Linguistics, Seattle.

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Training pruned neural networks. CoRR, abs/1803.03635.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. 2019. Stabilizing the lottery ticket hypothesis. CoRR, abs/1903.01611.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, pages 427–431, Valencia, Spain. Association for Computational Linguistics.

Marcin Junczys-Dowmunt. 2018. Microsoft's submission to the WMT2018 news translation task: How I learned to stop worrying and love the data. In Proceedings of the Third Conference on Machine Translation: Shared Task Papers, pages 425–430, Belgium, Brussels. Association for Computational Linguistics.

Marcin Junczys-Dowmunt. 2019. Microsoft translator at WMT 2019: Towards large-scale document-level neural machine translation. In Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1), pages 225–233, Florence, Italy. Association for Computational Linguistics.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, et al. 2018a. Marian: Fast neural machine translation in C++. In Proceedings of ACL 2018, System Demonstrations, pages 116–121.

Marcin Junczys-Dowmunt, Kenneth Heafield, Hieu Hoang, Roman Grundkiewicz, and Anthony Aue. 2018b. Marian: Cost-effective high-quality neural machine translation in C++. In Proceedings of the 2nd Workshop on Neural Machine Translation and Generation, pages 129–135.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1317–1327.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019. From research to production and back: Ludicrously fast neural machine translation. In Proceedings of the 3rd Workshop on Neural Generation and Translation, pages 280–288, Hong Kong. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 66–71.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Andres Rodriguez, Eden Segal, Etay Meiri, Evarist Fomenko, Young Jin Kim, Haihao Shen, and Barukh Ziv. 2018. Lower numerical precision deep learning inference and training.

O. Tange. 2011. Gnu parallel - the command-line power tool. ;login: The USENIX Magazine, 36(1):42–47.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.